+ Répondre à la discussion | Signaler un problème

Discussion : **Oubliez le « prompt engineering » ; le « loop engineering » fait désormais fureur**
Sujet : *Intelligence artificielle*

Outils de la discussion | Affichage

Aujourd'hui, 14h16 | #1

Mathis Lucas
Chroniqueur Actualités
Rédacteur technique
Inscrit en: Juin 2023
Messages: 1 933

Oubliez le « prompt engineering » ; le « loop engineering » fait désormais fureur

Oubliez le « prompt engineering » ; le « loop engineering » fait désormais fureur. Plutôt que de dicter chaque instruction à l'agent IA, vous concevez une boucle autonome qui s'en charge à votre place

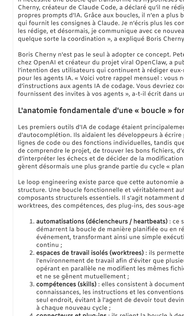
Le loop engineering est une évolution majeure de l'IA agentique. Il consiste à concevoir des systèmes autonomes récurrifs capables de s'autoplaniez, remplaçant ainsi l'interaction manuelle par des cycles itératifs de planification, d'action et de vérification. Une boucle efficace repose sur des piliers essentiels tels que les automatisations, l'isolation des tâches via des worktrees, et une mémoire externe sur disque servant de colonne vertébrale au projet. L'objectif est de permettre à l'IA d'agir, de vérifier ses erreurs via des outils de test et de corriger son propre code jusqu'à l'atteinte d'un but précis. Cependant, ce concept introduit des risques pour la sécurité.

Le loop engineering (ingénierie des boucles) consiste à concevoir, exploiter et améliorer les boucles de rétroaction qui permettent aux agents d'IA de codage de planifier leur travail, de modifier le code, d'observer les résultats et de réviser leur approche jusqu'à ce qu'une tâche logicielle soit achevée. Cela revient en effet à se substituer à soi-même en tant que personne qui donne des instructions à l'agent. On conçoit à la place un système qui s'en charge.

Plutôt que de considérer un outil d'IA comme un simple générateur de code ponctuel, le loop engineering adopte le développement logiciel comme un système itératif : définir l'objectif, inspecter la base de code, faire des modifications, effectuer une validation, analyser le résultat et décider de la prochaine étape.

L'évolution de l'ingénierie de l'IA vers le loop engineering

Selon les commentaires de plusieurs experts du secteur, l'émergence du loop engineering s'inscrit dans une progression logique de l'industrie en quatre étapes distinctes. L'approche initiale était l'ingénierie des prompts, qui a l'effort se concentrait sur la formulation précise des requêtes envoyées au modèle d'IA. Ensuite a suivi l'apparue l'ingénierie du contexte, qui visait à organiser et fournir à l'agent IA les bonnes informations au moment opportun de l'inférence.



La troisième étape fut l'ingénierie du harnais, permettant d'encapsuler l'agent dans un environnement d'exécution fiable doté d'outils spécifiques. Enfin, le loop engineering est la couche la plus récente et dynamique qui met cet ensemble en mouvement grâce à des cycles continus de raisonnement, d'action et d'évaluation.

Ce concept est important, car, dans l'ingénierie logicielle moderne, un simple prompt ne suffit que rarement à résoudre un problème. Les projets concrets comportent des contraintes cachées, des tests peu fiables, des conventions héritées, des règles de déploiement, des compromis liés au produit et des cahiers des charges incomplets. Un workflow de codage basé sur l'IA pour être utile ne se limite pas à un modèle capable de générer du code plausible.

Il nécessite une boucle qui transforme les hypothèses en progrès vérifié. Boris Cherny, créateur de Claude Code, a déclaré qu'il ne rédigeait plus beaucoup ses propres prompts d'IA. Grâce aux boucles, il n'en a plus besoin. « C'est un agent qui fournit les consignes à Claude. Je n'écris plus les consignes, c'est Claude qui les rédige, et désormais, je communique avec ce nouveau Claude qui assure en quelque sorte la coordination », a expliqué Boris Cherny.

Boris Cherny n'est pas le seul à adopter ce concept. Peter Steinberger, ingénieur chez OpenAI et créateur du projet viral OpenClaw, a publié un rappel à l'intention des utilisateurs qui continuent à rédiger eux-mêmes des prompts pour les agents IA. « Voici votre rappel mensuel : vous ne devriez plus donner d'instructions aux agents IA de codage. Vous devriez concevoir des boucles qui fournissent des invites à vos agents », a-t-il écrit dans un billet sur X.

L'anatomie fondamentale d'une « boucle » fonctionnelle

Les premiers outils d'IA de codage étaient principalement des systèmes d'autocomplétion. Ils aidaient les développeurs à écrire plus rapidement des lignes de code ou des fonctions individuelles, tandis que l'humain restait chargé de comprendre le projet, de trouver les bons fichiers, d'exécuter les tests, d'interpréter les échecs et de décider de la modification suivante. Les agents IA gèrent désormais une plus grande partie du cycle « planifier-agir-vérifier ».

Le loop engineering existe parce que cette autonomie accrue a besoin d'une structure. Une boucle fonctionnelle et véritablement autonome repose sur six composants structurels essentiels, il s'agit notamment du déclencheur, des workflows, des dépendances, des plug-ins, des sous-agents et une mémoire :

1. **automatisations (déclencheurs / heartbeats)** : ce sont elles qui démarrent la boucle de manière planifiée ou en réponse à un événement, transformant ainsi une simple exécution en un cycle continu ;
2. **espaces de travail isolés (worktrees)** : ils permettent d'isoler l'environnement de travail afin d'éviter que plusieurs agents opèrent en parallèle ne modifient les mêmes fichiers simultanément et ne se gênent mutuellement ;
3. **compétences (skills)** : elles consistent à documenter les connaissances, les instructions et les conventions du projet en un seul endroit, évitant à l'agent de devoir tout deviner ou réapprendre à chaque nouveau cycle ;
4. **connecteurs et plug-ins** : ils relient la boucle à des outils externes (comme GitHub, Slack ou Linear), ce qui permet à l'agent d'agir réellement dans votre environnement de travail (par exemple, ouvrir une Pull Request), plutôt que de seulement suggérer des modifications de fichiers ;
5. **sous-agents (sub-agents)** : ils permettent de séparer les rôles en divisant l'agent qui crée le code de l'agent qui l'examine et l'approuve, car un modèle évaluant son propre travail a tendance à manquer de rigueur ;
6. **l'état ou la mémoire (state / memory)** : souvent appelé la colonne vertébrale du système, il s'agit d'un fichier externe (comme un fichier Markdown) qui enregistre ce qui a été fait et ce qu'il reste à accomplir, car les modèles d'IA oublient tout leur contexte entre chaque exécution.

À en croire les rapports d'expérience partagés en ligne, le loop engineering s'adapte aux besoins spécifiques du développement grâce à plusieurs modèles d'exécution. On retrouve notamment des boucles axées sur les tests, où l'agent modifie le code en boucle jusqu'à ce qu'un test spécifique réussisse, ou encore des boucles guidées par le compilateur, qui s'avèrent redoutables pour résoudre des erreurs structurelles lors de migrations de code.

D'autres approches se basent sur l'inspection de l'exécution pour le débogage en temps réel ou intègrent directement les retours des réviseurs humains. Une règle de conception primordiale est l'établissement de conditions d'arrêt vérifiables et plafonnées. Sans un nombre maximum d'essais, une limite de temps ou de budget, une boucle défectueuse tentera indéfiniment d'accomplir une tâche impossible, générant ainsi des coûts astronomiques.

Idéalement, l'arrêt de la boucle doit être dicté par la confirmation de réussite d'un outil externe, comme un exécuteur de tests, et non par l'auto-évaluation du modèle qui vient de rédiger le code. Par ailleurs, comme tout ce qui concerne les agents IA, cette approche de codage comporte également des risques.

Les risques et dérives liés à cette automatisation itérative

Malgré son efficacité apparente, la mise en place de boucles autonomes comporte des vulnérabilités complexes à gérer. La dérive du contexte est un risque majeur : au fil d'une longue session, l'historique s'accumule, saturant la mémoire de travail de l'agent qui finit par s'appuyer sur des hypothèses obsolètes. Le phénomène de piétinement est aussi fréquent lorsque la boucle modifie sans cesse des fichiers sans réussir à converger vers une solution.

De plus, le risque de surajustement aux tests peut amener l'agent à modifier le code uniquement pour satisfaire techniquement le validateur, tout en détruisant le comportement attendu par l'utilisateur final. Le coût financier caché demeure l'une des erreurs les plus dommageables dans ce type d'approche, car la multiplication silencieuse des itérations et des appels aux différents modèles peut rapidement faire exploser la consommation de tokens.

Enfin, l'adoption du loop engineering transforme le rôle du développeur, mais ne le fait en aucun cas disparaître. L'effort humain est déplacé vers les deux extrémités du processus : la définition extrêmement précise des intentions au démarrage et la responsabilité finale lors de la validation du code produit.

L'une des menaces les plus pernicieuses d'une boucle bien huilée est l'érosion de la compréhension du projet par le développeur lui-même. Ce dernier est tenté d'approuver le code sans l'analyser, un piège qualifié de "renoncement cognitif". Le développeur doit rester activement impliqué : examiner les différences générées et utiliser cette technologie pour accélérer le travail tout en assumant la pleine responsabilité de la qualité finale du logiciel livré.

Conclusion

Pour l'instant, le débat sur les boucles porte principalement sur le codage agentique. Cela ne signifie pas pour autant que les boucles soient réservées aux ingénieurs en informatique. « C'est le moment pour les managers d'entrer en scène », a déclaré VO. « Vous concevez un poste. Imaginez donc que vous intégriez un nouvel employé. Cet employé pourrait être un assistant de direction, un agent au service client ou un ingénieur en informatique ».

La principale préoccupation cible concernant les boucles est, de loin, leur coût. Faire tourner plusieurs agents avec des sous-agents sur le tout dernier modèle d'IA de pointe est un excellent moyen d'épuiser votre budget personnel de tokens ou d'attirer les regards méfiants de votre patron.

Lorsque des utilisateurs sur X ont demandé à Peter Steinberger comment il modifierait sa boucle pour être plus économe en tokens, il a répondu : « se réveiller et effectuer quelques appels API coûte relativement peu cher, ou bien opter pour une fois par heure/jour pour réduire la consommation de tokens ». Dans un billet de blogue sur le sujet, Addy Osmani, directeur de Google Cloud, conseille de ne dépenser que lorsque cela est nécessaire.

Sources : billet de blogue, Peter Steinberger

Et vous ?

➡ Quel est votre avis sur le sujet ?
➡ Que pensez-vous du nouveau concept de loop engineering ?
➡ Va-t-il permettre d'améliorer l'efficacité de l'IA en matière de codage ?
➡ Que pensez-vous des préoccupations en matière de coût liées au loop engineering ?

Voir aussi

➡ Description technique détaillée du fonctionnement interne de l'agent de codage d'OpenAI Codex CLI, un outil de codage IA qui écrit du code, exécute des tests et corrige des bogues

➡ Le piège du codage par l'IA, par Chris Lov

➡ « Les développeurs recourent systématiquement au modèle le plus puissant, même pour des tâches triviales. C'est comme prendre sa Ferrari pour acheter du lait », affirme Neel Sundaresan, ingénieur chez IBM

Répondre avec citation | 6 | 0

Aujourd'hui, 14h42 | #2

PomFritz
Membre confirmé
VIP
Inscrit en: Mai 2008
Messages: 220

L'important, c'est de vendre du rêve. Comme on dit "un singe tapant indéfiniment et au hasard sur un clavier finira par écrire n'importe quel texte donné, y compris l'œuvre complète de Shakespeare."

Répondre avec citation | 1 | 0

Aujourd'hui, 15h47 | #3

smarties
Membre éprouvé
Développeur
Inscrit en: Août 2003
Messages: 1 556

> Envoyé par **Mathis Lucas**
> *Enfin, l'adoption du loop engineering transforme le rôle du développeur, mais ne le fait en aucun cas disparaître. L'effort humain est déplacé vers les deux extrémités du processus : la définition extrêmement précise des intentions au démarrage et la responsabilité finale lors de la validation du code produit.*

Quand je vois des MR de plus de 1000 lignes de modification il est parfois difficile de bien se rappeler de tout donc j'imagine la validation du code produit sur un projet complet : 100k+ de lignes de code d'un coup

Je préfère continuer ce que je fais, au niveau macro, je lui fait rédiger une roadmap avec chaque item numéroté et bien décrit, puis pour je fais plus ou moins les prompts pour chaque élément (a la fin de son travail, il est sensé me générer des test et me proposer un conventional commit). Je fais la revue, pose des questions, le corrige avant de commit. Comme ça j'ai rarement plus de 100 lignes mêter d'un coup, la plus grosse partie concerne souvent les tests.

Répondre avec citation | 0 | 0

Responsable Domaine Si Hf
↳ CREDIT AGRICOLE PROTECTION & SÉCURITÉ - Pays de la Loire - Coulaines (72190)

Experte / Expert produit éditique
↳ LA BANQUE POSTALE - France - Gradignan

Voir plus d'offres

Discussions similaires

Le prompt engineering, qui a émergé avec l'essor de l'IA générative, est en voie d'extinction
Par Anthony dans le forum Intelligence artificielle
Réponses: 7
Dernier message: 14/05/2025, 10h38

[Livre] Prompt Engineering for Generative AI - Future-Proof Inputs for Reliable AI Outputs
Par dourouc05 dans le forum Livres
Réponses: 0
Dernier message: 15/12/2024, 19h00

Reverse engineering dans Looping
Par gloriaID dans le forum Looping
Réponses: 4
Dernier message: 22/07/2023, 00h25

Google fait l'acquisition de la startup Urban Engines qui exploite le big data pour optimiser le transport
Par Stéphane le calme dans le forum Actualités
Réponses: 3
Dernier message: 20/09/2016, 13h57

Tutoriels et liens pour le Borland Database Engine
Par Community Management dans le forum Paradox
Réponses: 0
Dernier message: 25/03/2002, 10h23

Partager